# Lab 3: Deep learning

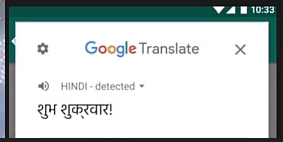
ELMED219 – 2023

Friday, January 20th, 2023

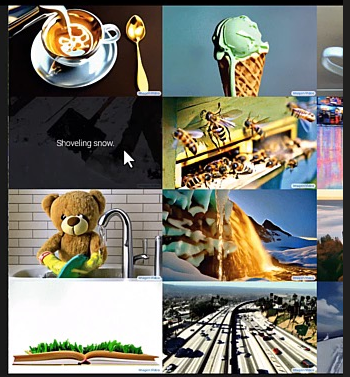
Western Norway University of Applied Sciences

Alexander S. Lundervold, lundervold.net

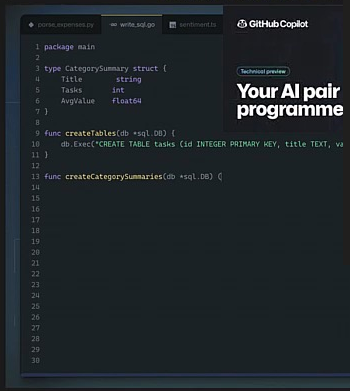


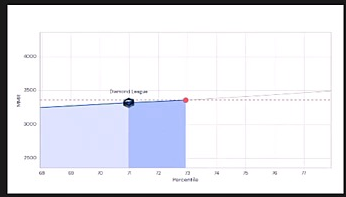



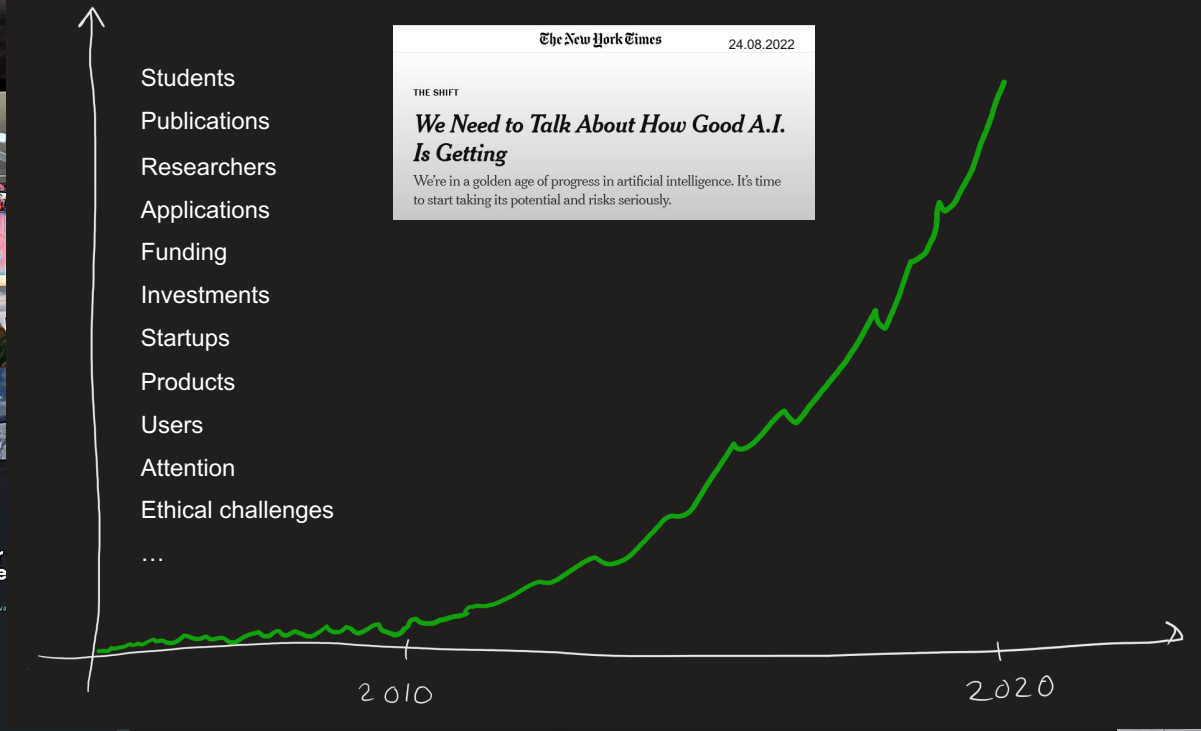






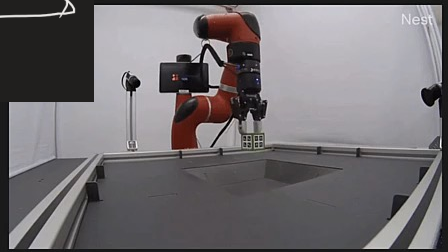






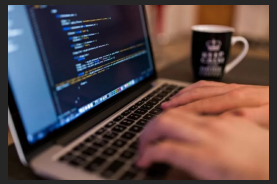





ChatGPT


*Attributions for illustrations used in the presentation are available upon request*

# Plan



Hands-on

What *is* deep learning?

How do you *do* deep learning?

Recent developments in generative AI

## Deep learning

*searching for good hierarchical geometric representations*

## Deep learning

*searching for good hierarchical geometric representations*

## Deep learning

*searching for good hierarchical geometric representations*

## Deep learning

*searching for good hierarchical geometric representations*

## Deep learning

*searching for good hierarchical geometric representations*

01

02

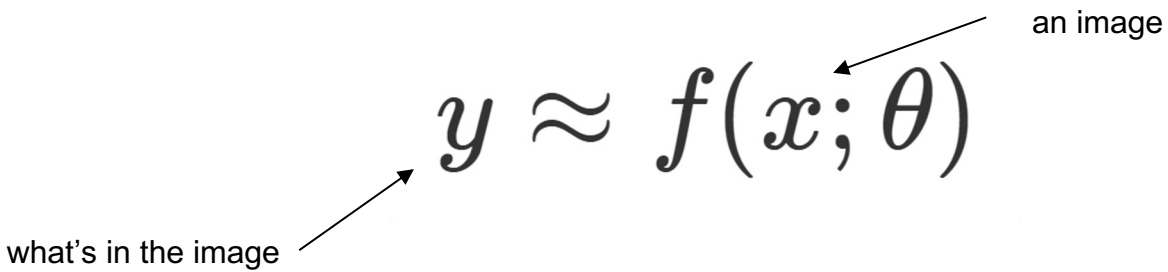
03

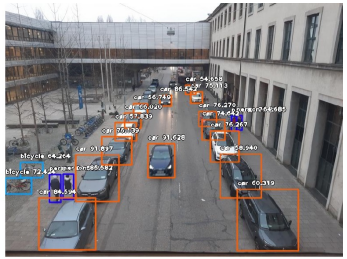
04

05

# Function approximation

Function approximation





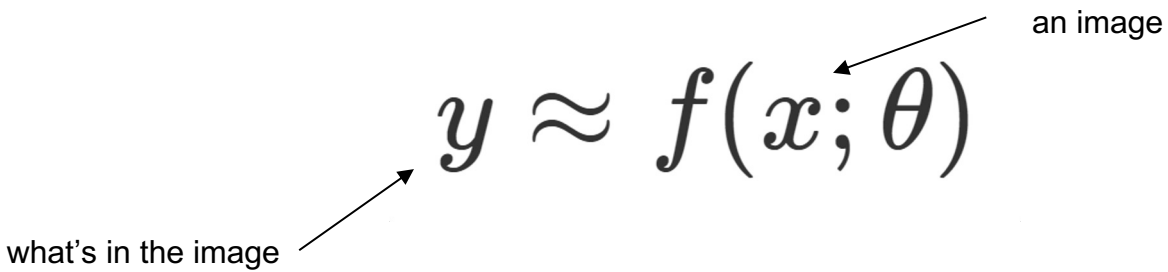
Function approximation

an image

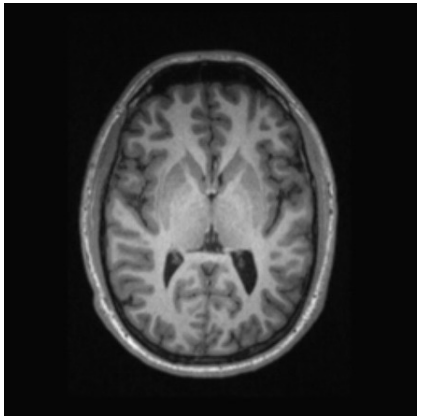
$$y \approx f(x;\theta)$$

what's in the image

Function approximation



AGE

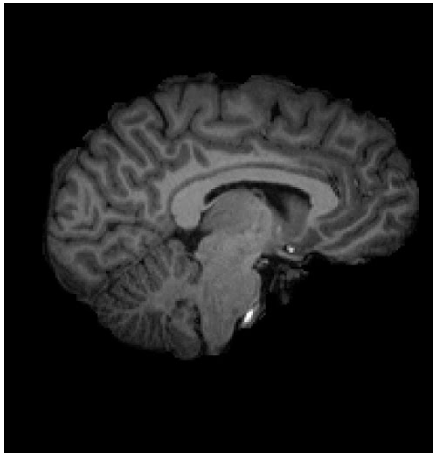


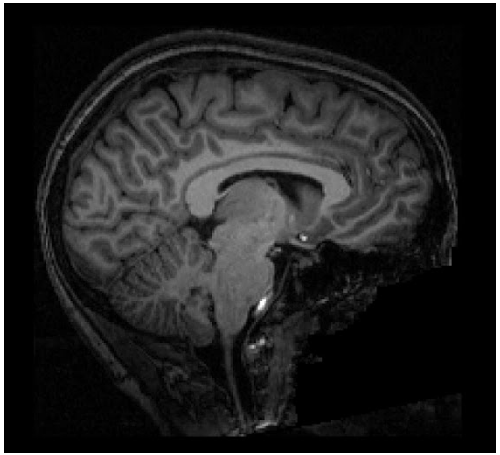
Function approximation

an MRI image

$$y \approx f(x;\theta)$$

”Brain age”



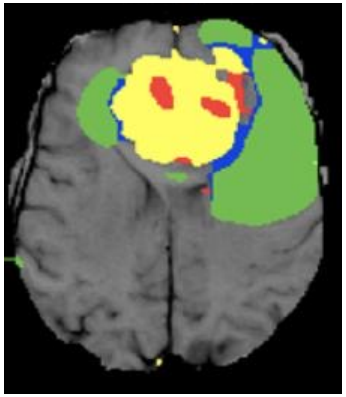


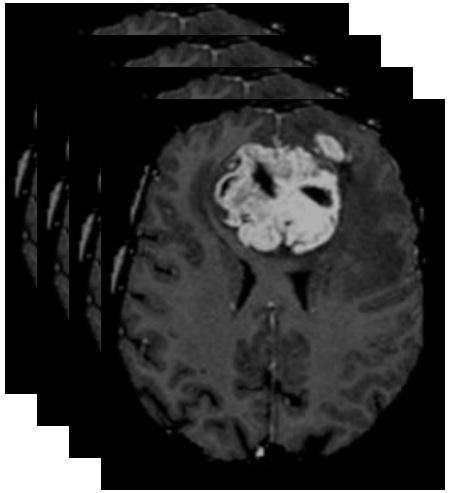
Function approximation

a brain MRI

$$y \approx f(x;\theta)$$

skull-stripped MRI



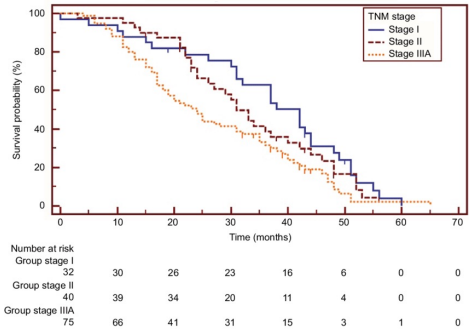


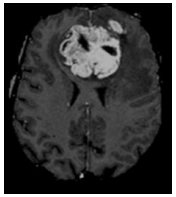
Function approximation

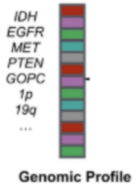
multimodal MRI images

$$y \approx f(x;\theta)$$

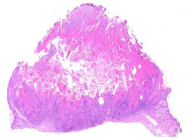
Segmentation of region-of-interests (e.g., tumors)

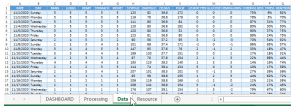





Genomic Profile





Function approximation

heterogeneous information

$$y \approx f(x;\theta)$$

outcome (e.g., survival)

Text-to-speech

Speech-to-speech

Text-to-image

Time-series to anomaly yes/no

and much more…

Function approximation

$$y \approx f(x; \theta)$$

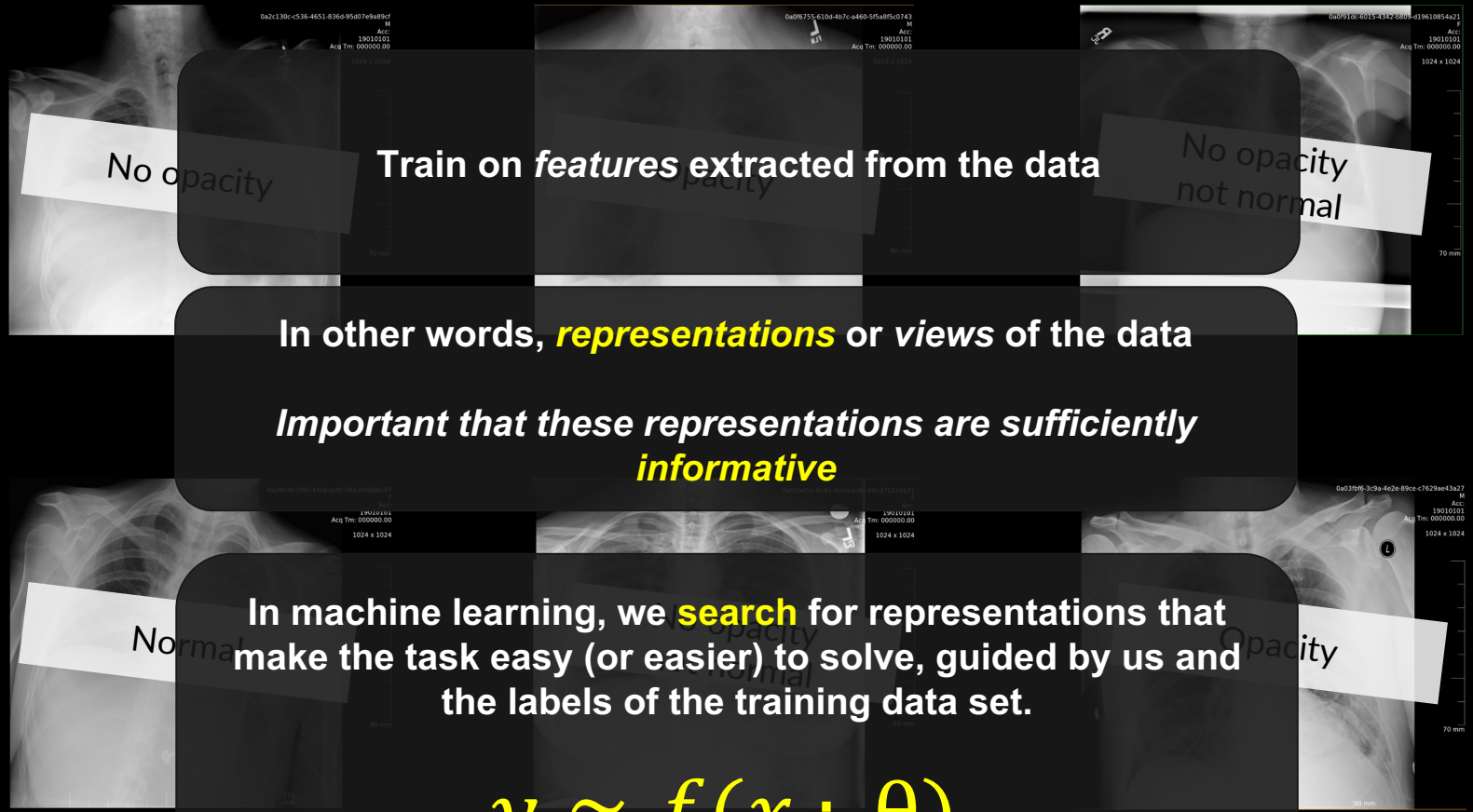
Training

$$(X, y)$$

Training

$$(X, y)$$


*This is the setup in what's called **supervised learning***




Train on *features* extracted from the data

In other words, ***representations*** or *views* of the data

***Important that these representations are sufficiently informative***

In machine learning, we **search** for representations that make the task easy (or easier) to solve, guided by us and the labels of the training data set.

$$y \approx f(x\,;\,\theta)$$

# 01 Machine learning: **function approximation** based on **training**

features of a disease, process, patient or system

$$y \approx f(x\,;\,\theta)$$

healthy or not healthy / outcome

Models and parameters

$$y \approx f(x; \theta)$$

**model** **parameters**

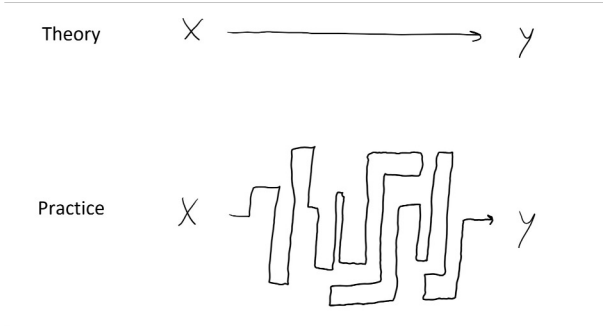
***Training*** **is a search for useful representations through a space defined by a family $\mathcal{F}$ of parametrized models**

**01** Machine learning: **function approximation** based on **training**

**02** We pick the model family $F$. The parameters $\Theta$ are found automatically through training

$$y \approx f(x;\theta)$$

# Theory versus practice




**In theory:**

can get arbitrarily good approximations by training simple models
(*universal approximation* and *no free lunch theorems*)

**In practice:**

which **f** we use and how the training is done makes a huge difference

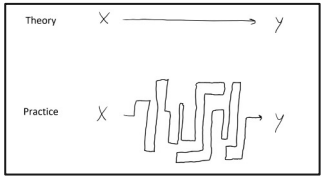
The family $\mathcal{F}$ of models considered, i.e., the *hypothesis space,* should be
(i) efficiently searchable, (ii) leading to expressive models that can
(iii) generalize beyond the training data distribution (i.e., work well on new data)

**01** Machine learning: **function approximation** based on **training**

**02** We pick the model family $F$. The parameters $\Theta$ are found automatically through training

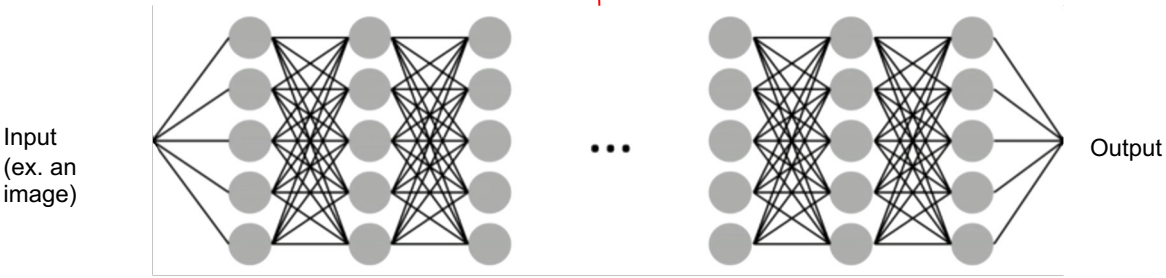
**03** In practice: some models are better suited to some tasks than others.

$$y \approx f(x;\theta)$$



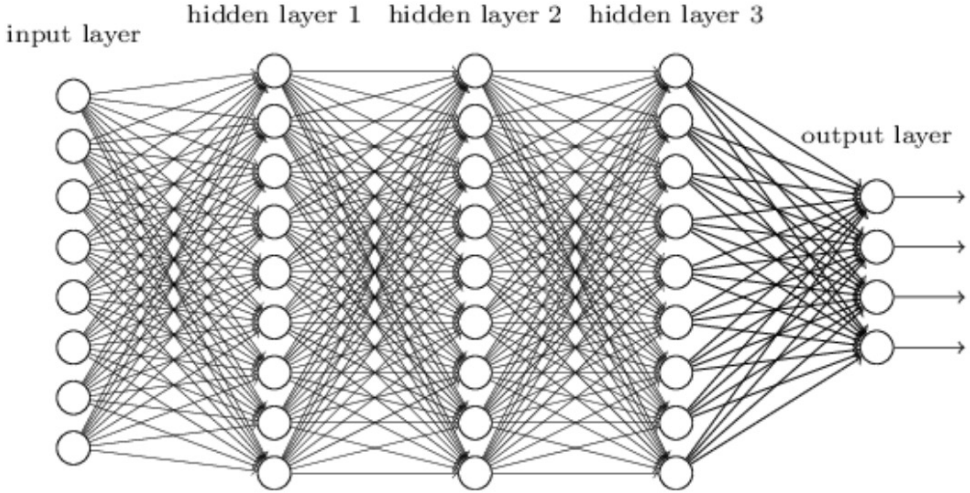
Deep learning

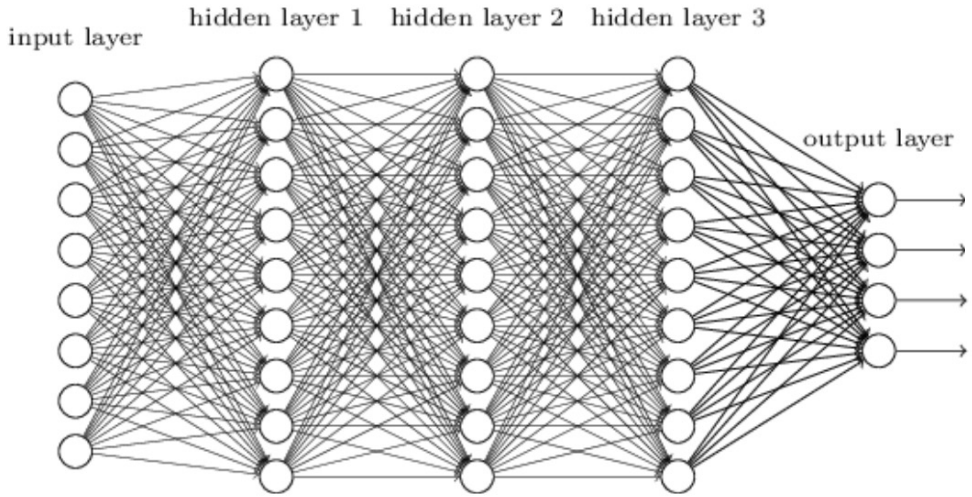
$$y \approx f(x; \theta)$$




**Artificial neural networks**

Computational graphs of simple units (“neurons”) connected in various specific ways, parametrized by parameters associated to each layer.


input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer




A geometric analogy: uncrumpling paper

This is a fancy way to draw what's really just a composition of simple functions:

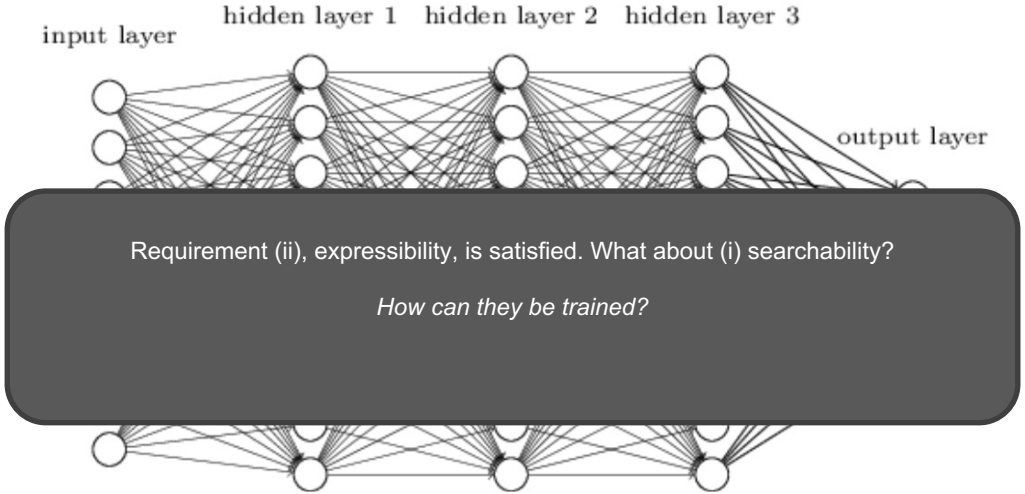
$$y = \sigma_{k+1}(W_{k+1}^T \sigma_k(W_k^T \left( \cdots \left( \sigma_1(W_1^T x) \right) \cdots \right) = f(x;\theta)$$

Input x is sent through several consecutive **layers** (functions). An output is produced.

Think of each layer as a (geometric) **transformation** of its input.

Every layer has a bunch of **parameters**, stored in the matrices W

*Each family member of F is complex transformation broken down into a composition of simpler ones, parametrized by the layer weights.*




Requirement (ii), expressibility, is satisfied. What about (i) searchability?

*How can they be trained?*

This is a fancy way to draw what's really just a composition of simple functions:

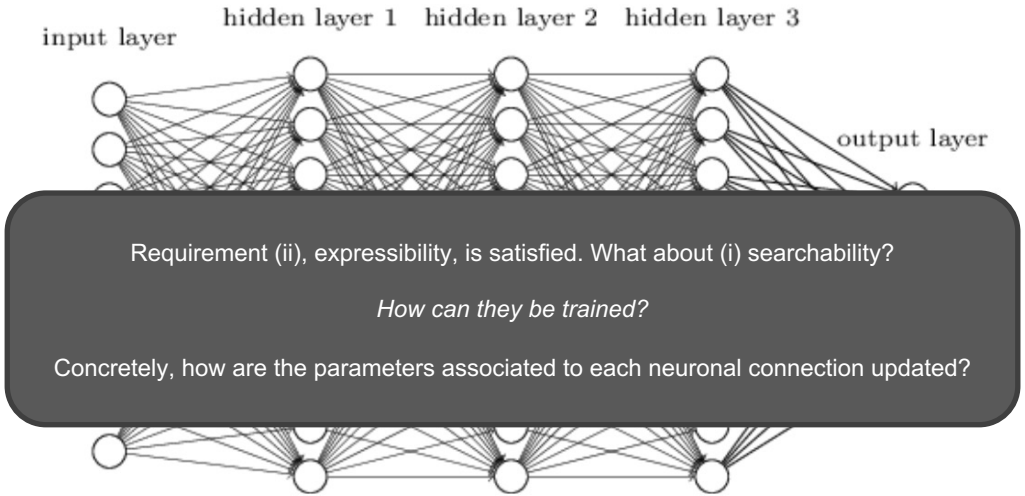
$$y = \sigma_{k+1}(W_{k+1}^T \sigma_k(W_k^T \left( \cdots \left( \sigma_1(W_1^T x) \right) \cdots \right) = f(x;\theta)$$

Input x is sent through several consecutive **layers** (functions). An output is produced.

Think of each layer as a (geometric) **transformation** of its input.

Every layer has a bunch of **parameters**, stored in the matrices W

*Each family member of F is complex transformation broken down into a composition of simpler ones, parametrized by the layer weights.*




Requirement (ii), expressibility, is satisfied. What about (i) searchability?

*How can they be trained?*

Concretely, how are the parameters associated to each neuronal connection updated?

This is a fancy way to draw what's really just a composition of simple functions:

$$y = \sigma_{k+1}(W_{k+1}^T \sigma_k(W_k^T \left( \cdots \left( \sigma_1(W_1^T x) \right) \cdots \right) = f(x;\theta)$$

Input x is sent through several consecutive **layers** (functions). An output is produced.

Think of each layer as a (geometric) **transformation** of its input.

Every layer has a bunch of **parameters**, stored in the matrices W

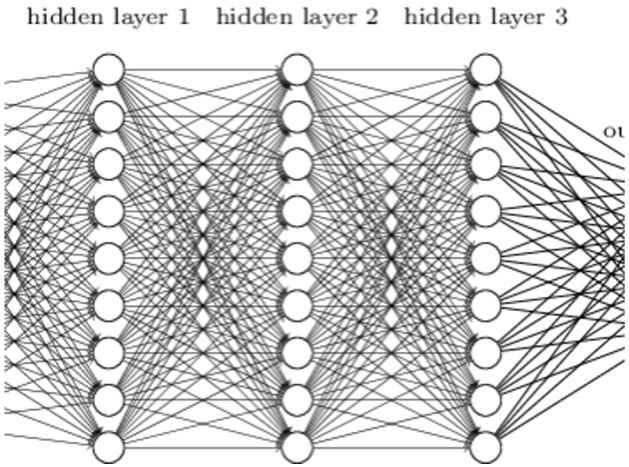
*Each family member of F is complex transformation broken down into a composition of simpler ones, parametrized by the layer weights.*

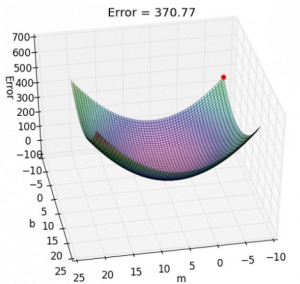
# Three basic ingredients:

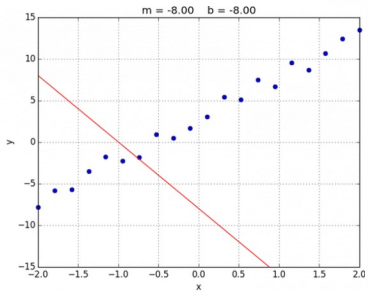
1. A **loss function**: Used to measure the network's performance on the training data. A **feedback signal** for the training process.

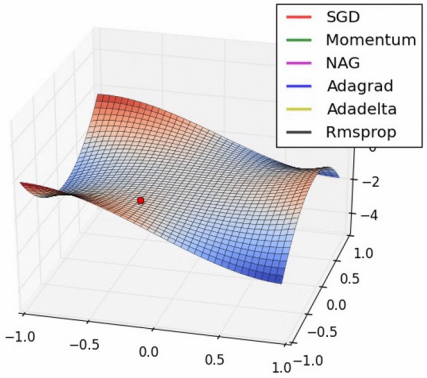
2. An **optimizer**: Used to update the *parameters* of the network to increase performance as measured by the loss function. **Gradient descent** and **backpropagation**.

3. One or more **metrics**: A way to score the model. For example *accuracy* or *mean  squared error.*



Define a neural network model with an architecture and a set of trainable parameters (weights and biases)

1. Initialize the parameters

2.Collect a batch of inputs `(X,y)` and use the model with its current parameters to make predictions `y_pred`

3. Calculate how good the model is using the loss function to measure the discrepancy between the known labels `y_true` and the predictions `y_pred` for this batch

4. Calculate the gradients for each parameter using backpropagation. This measures how changing each parameter would change the loss. In other words, each parameter's contribution to the loss.

5. Move each weight a little bit in the opposite direction of its gradient using gradient descent. I.e., some variant of

```
parameter = parameter - learning_rate * gradient
```

6. Go back to step 2 and repeat the process

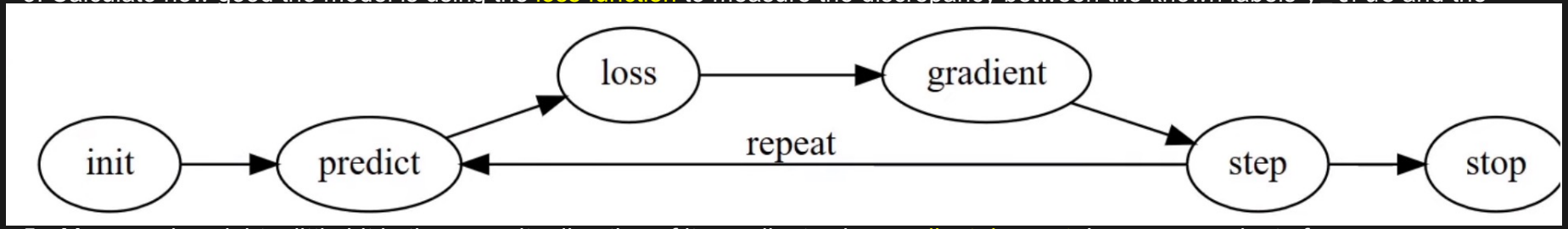
7. Do this over and over for several epochs (one epoch is one pass through all the training data), until you meet some chosen *stopping criteria*

Define a neural network model with an architecture and a set of trainable parameters (weights and biases)

1. Initialize the parameters

2.Collect a batch of inputs `(X,y)` and use the model with its current parameters to make predictions `y_pred`

3. Calculate how good the model is using the loss function to measure the discrepancy between the known labels `y_true` and the




5. Move each weight a little bit in the opposite direction of its gradient using gradient descent. I.e., some variant of
   `parameter = parameter - learning_rate * gradient`

6. Go back to step 2 and repeat the process

7. Do this over and over for several epochs (one epoch is one pass through all the training data), until you meet some chosen *stopping criteria*

Define a neural network model with an architecture and a set of trainable parameters (weights and biases)

1. Initialize the parameters

2.Collect a batch of inputs `(X,y)` and use the model with its current parameters to make predictions `y_pred`

3. Calculate how good the model is using the loss function to measure the discrepancy between the known labels `y_true` and the predictions `y_pred` for this batch

**What is gradient descent?**

4. Calculate the gradients for each parameter using backpropagation. This indicates how changing each parameter would change the loss. In other words, each parameter's contribution to the loss.

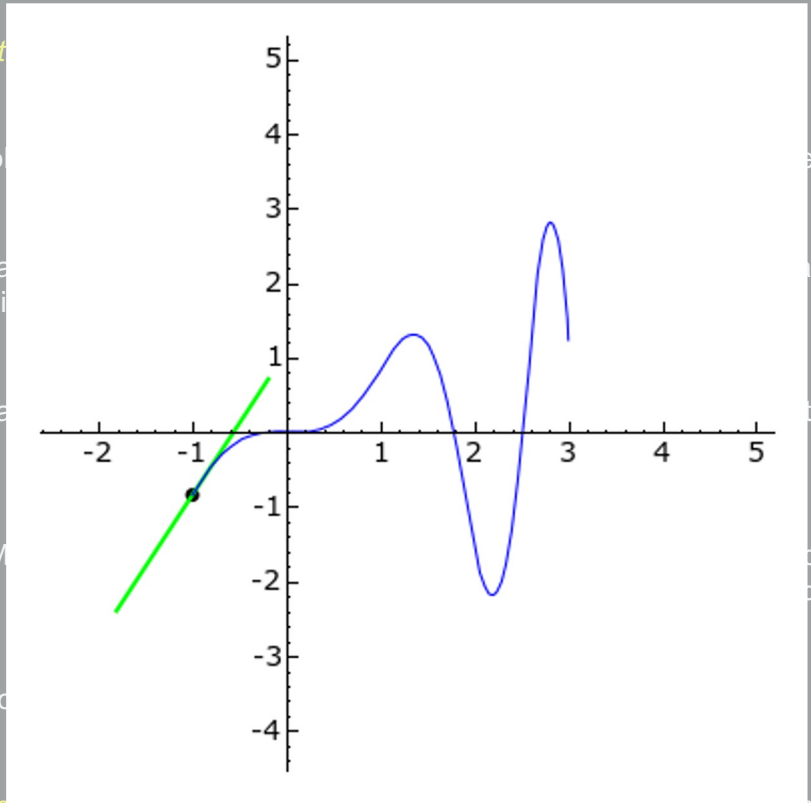
5. Move each weight a little bit in the opposite direction of its gradient using gradient descent. I.e., some variant of

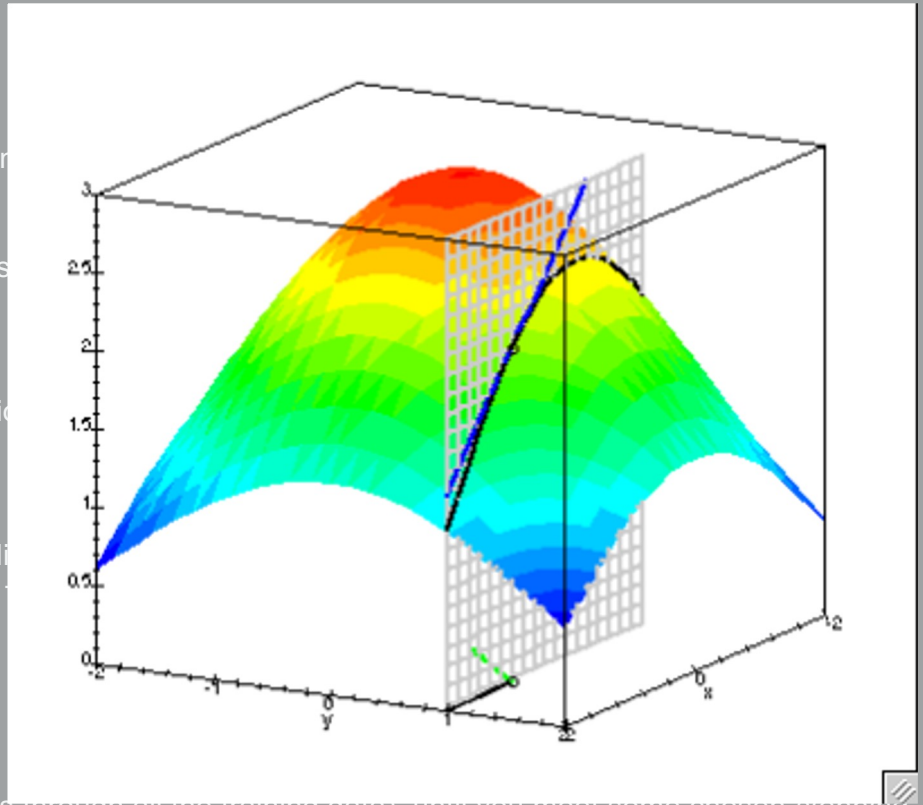
```
parameter = parameter - learning_rate * gradient
```

6. Go back to step 2 and repeat the process

7. Do this over and over for several epochs (one epoch is one pass through all the training data), until you meet some chosen *stopping criteria*

Define a neural network model with an architecture and a set of trainable parameters (weights and biases)




*1.Init*

2.Co

3. Ca
predi

4. Ca
loss.

5. M

6. Go

7. Do this over and over for several epochs (one epoch is one pass through all the training data), until you meet some chosen *stopping criteria*

Define a neural network model with an architecture and a set of trainable parameters (weights and biases)

*1.Init*

2.Co

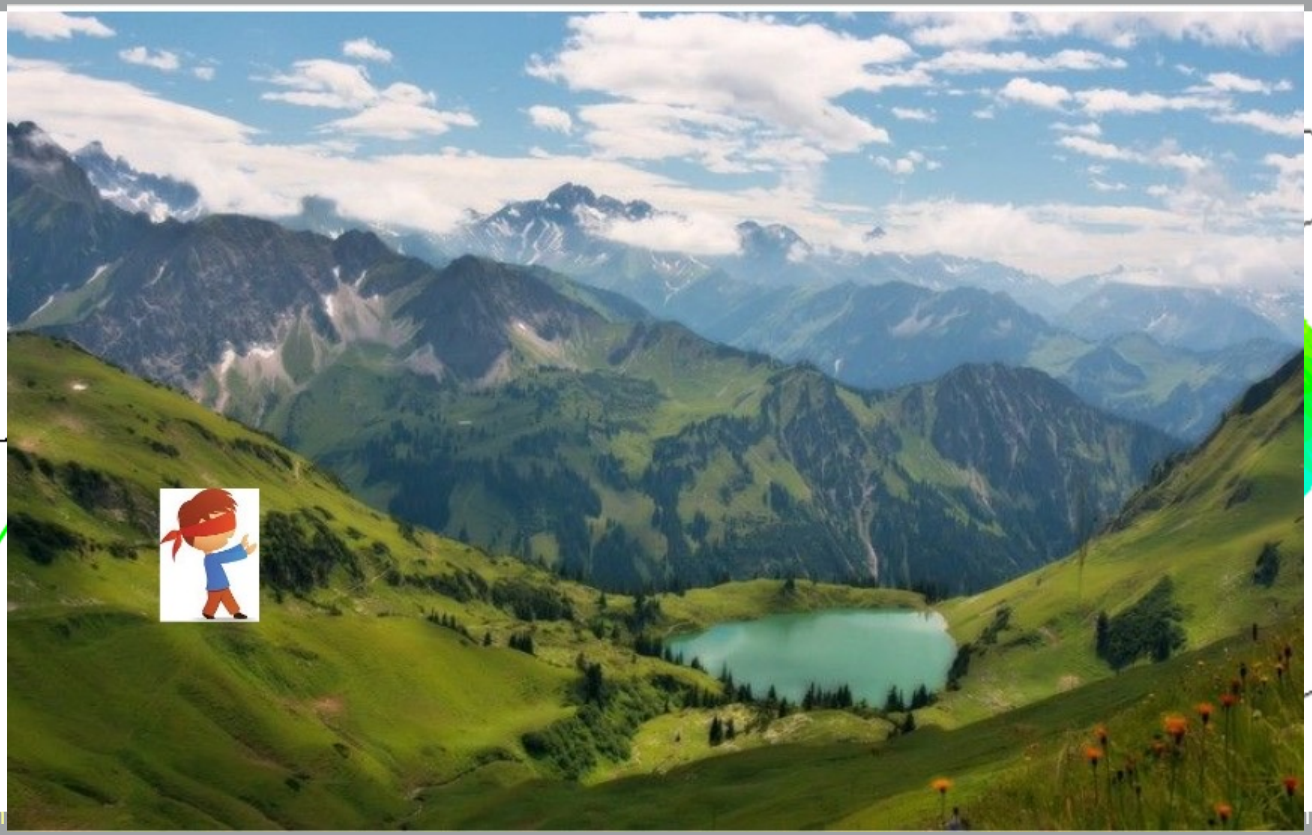
3. Ca
predi

4. Ca
loss.

5. M

6. Go

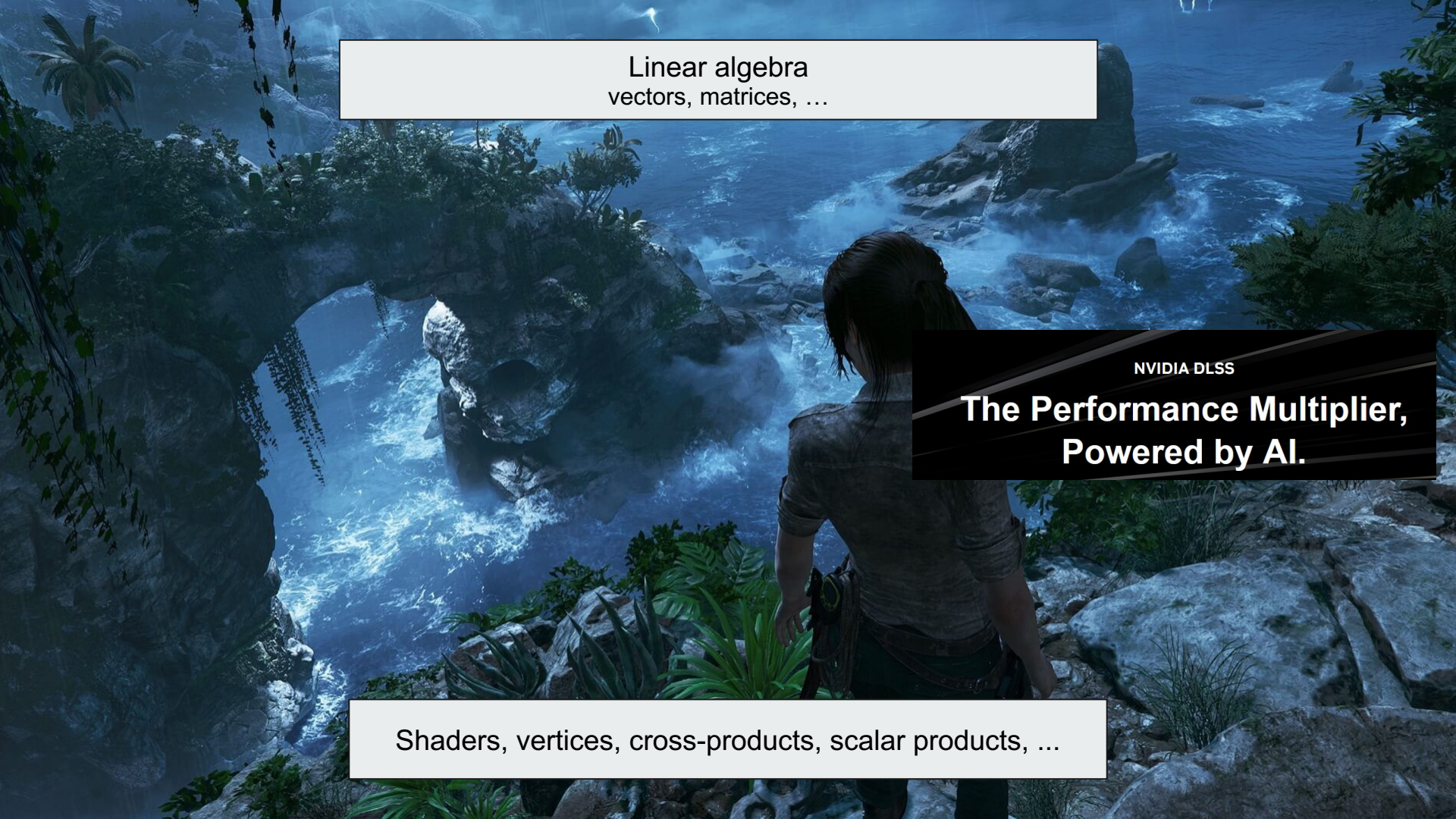
7. Do this over an… …e chosen *stopping criteria*




GPUs: *Massively parallel linear algebra super-computers*

Developed for computer graphics

*Good at exactly what's needed for neural networks!*


Linear algebra
vectors, matrices, …
NVIDIA DLSS
The Performance Multiplier,
Powered by AI.
Shaders, vertices, cross-products, scalar products, ...

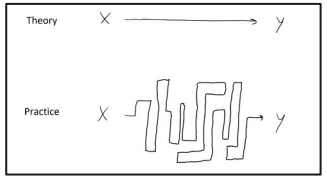
**01** Machine learning: **function approximation** based on **training**

**02** We pick the model family $\mathcal{F}$. The parameters $\Theta$ are found automatically through training

**03** In practice: some models are better suited to some tasks than others.

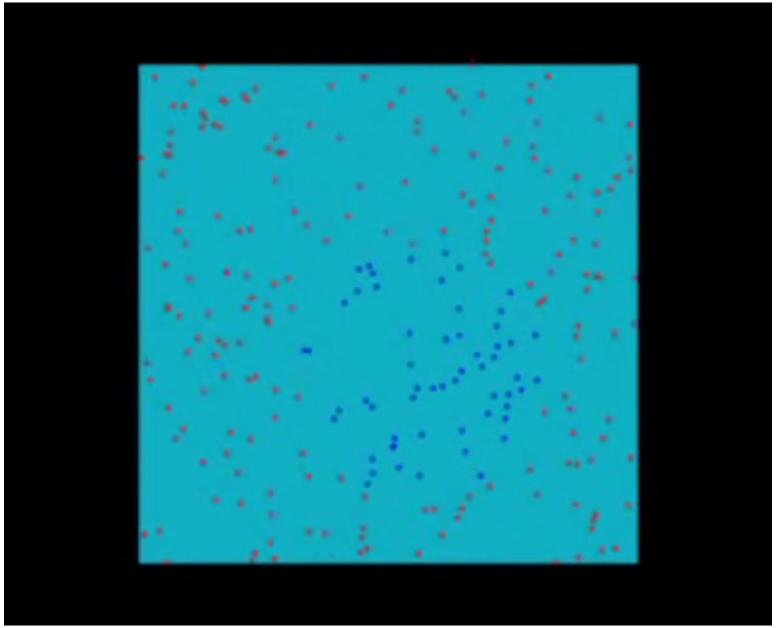
**04** In deep learning: a particular choice of $\mathcal{F}$. A **composition** of simple functions, trained jointly, typically using GPUs or other accelerators.

$$y \approx f(x;\theta)$$

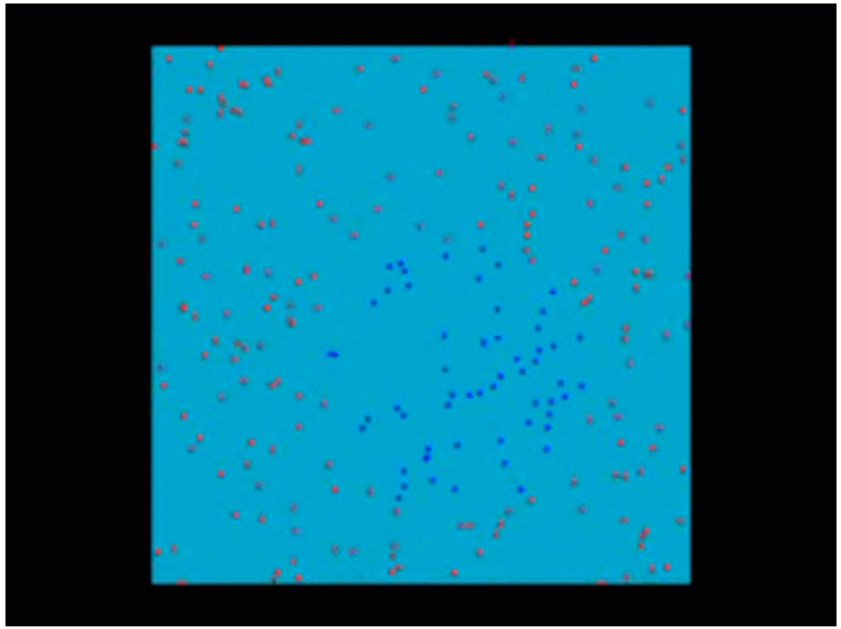



Representations and the power of transformations

# Transformations

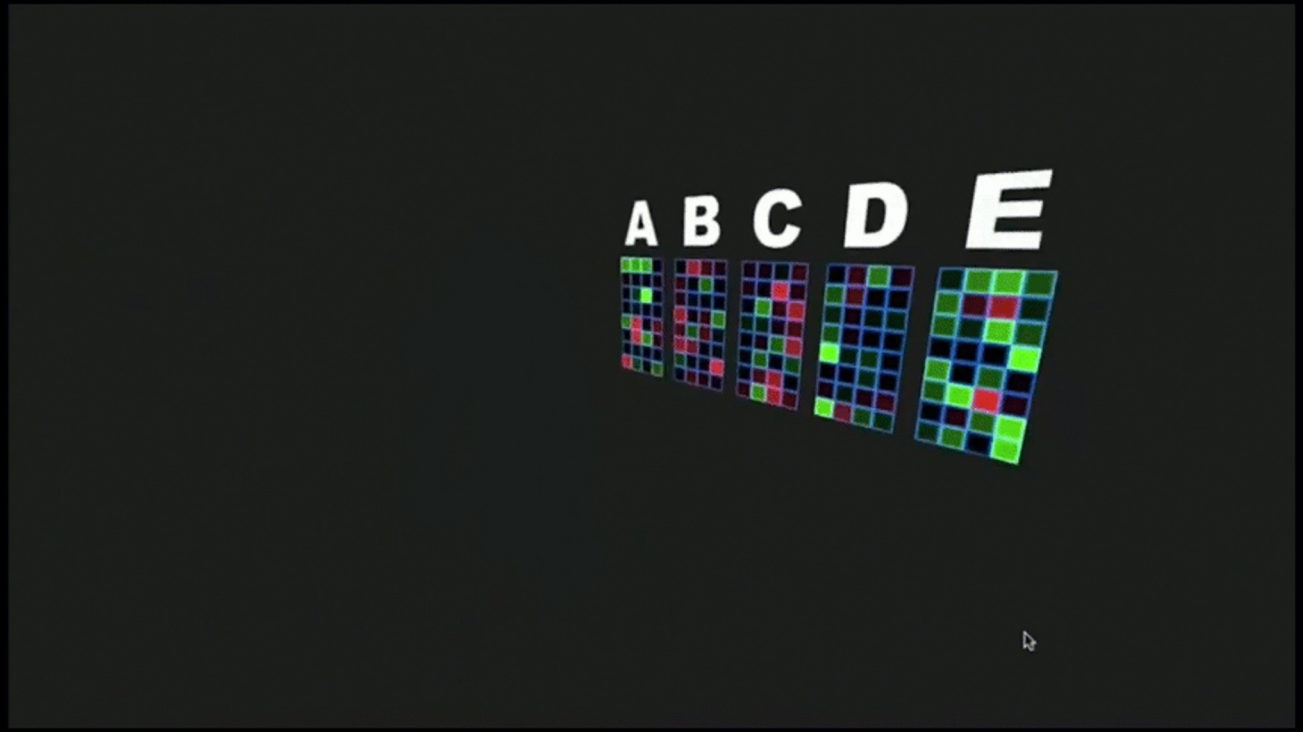


# Transformation

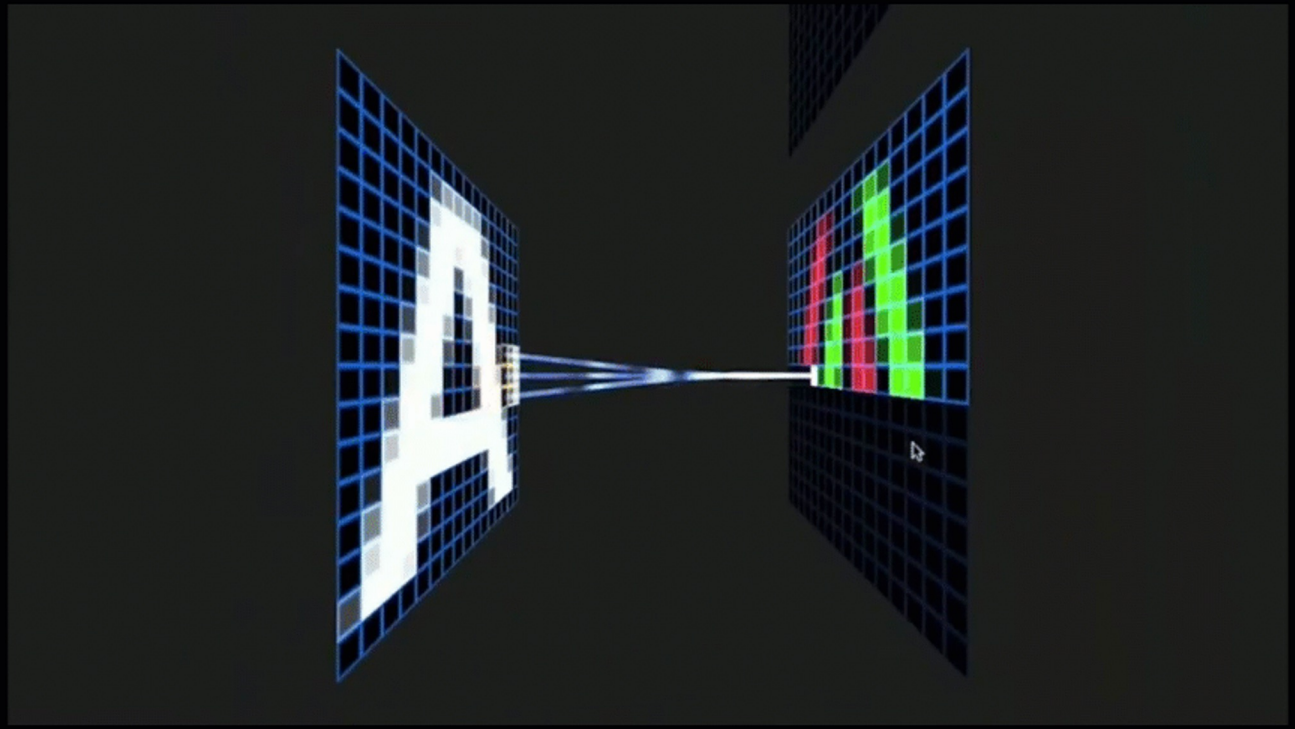


What transformations should be applied?

In deep learning: ***transformations providing useful representations automatically*** **found via training**




Source: Otavio Good



Source: Otavio Good

**01** Machine learning: **function approximation** based on **training**

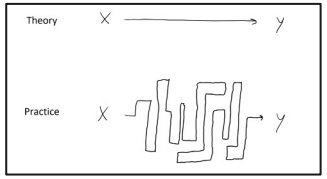
**02** We pick the model family $\mathcal{F}$. The parameters $\Theta$ are found automatically through training

**03** In practice: some models are better suited to some tasks than others.

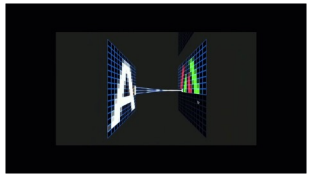
**04** In deep learning: a particular choice of $\mathcal{F}$. A **composition** of simple functions, trained jointly, typically using GPUs or other accelerators.

**05** In deep learning: finding a **hierarchical set of geometric transformations**, ending in a representation that makes the desired task easy to solve.

$$y \approx f(x;\theta)$$







**01** Machine learning: **function approximation** based on **training**

**02** We pick the model family $\mathcal{F}$. The parameters $\Theta$ are found automatically through training

**03** In practice: some models are better suited to some tasks than others.

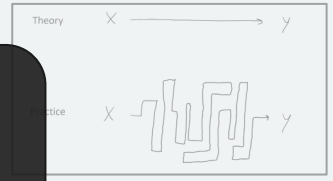
**04** In deep learning: a particular choice of $\mathcal{F}$. A **composition** of simple functions, trained jointly, typically using GPUs or other accelerators.

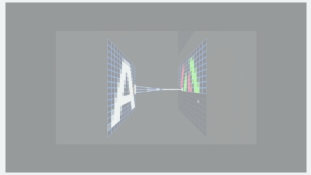
**05** In deep learning: finding a **hierarchical set of geometric transformations**, ending in a representation that makes the desired task easy to solve.

## Deep learning

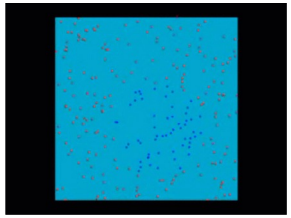
***searching for good hierarchical geometric representations***

$$y \approx f(x;\theta)$$

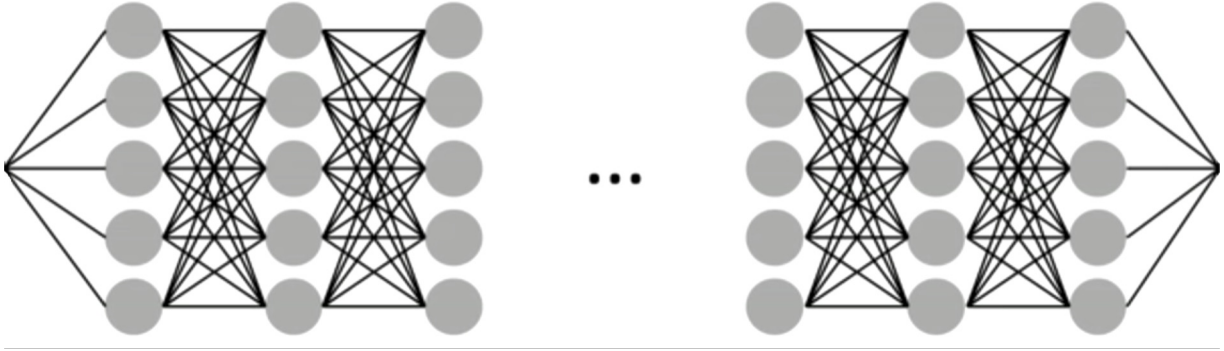


What distinguishes deep learning from other kinds of machine learning?



Input
(ex image)



Output

**Deep learning**
Artificial neural network

**“Classical” machine learning**

## In practice, this is a bit of a lie

**(albeit a useful lie, somewhat true)**


**Deep learning**

Artificial neural network

# Plan

What *is* deep learning?

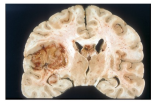
How do you *do* deep learning?

Recent developments in generative AI

jupyter 1.0-asi-brain_tumor_analysis_radiomics Last Checkpoint: an hour ago (unsaved changes)

A.S. Lundervold, 12.09.22

# Using deep learning to extract imaging-biomarkers for brain cancer analyses in MRI of glioblastoma

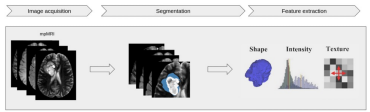


In this notebook, we'll use a deep learning model to segment brain tumors from multi-parametric MR and then extract features from the resulting tumor masks. Such features can potentially be associated with tumor severity and prognosis and contribute to better treatment.

Extracting features from objects of interest in medical images for diagnostic purposes is often referred to as **radiomics**. The goal of radiomics is to extract information from medical images that can be used as part of a medical imaging-based diagnostic workflow. The information can be extracted from various imaging modalities, e.g., different MRI contrasts, PET imaging, CT imaging, and so on. One can then combine it with other sources of information (e.g., demographics, clinical data, genetics). In such a way, radiomics-and **radiogenomics**-can open the door to sophisticated and powerful analyses.

## Radiomics workflow

In this notebook, we will estimate the locations and extent of the brain tumors' T2-enhancing and non-enhancing regions. We can then use this to extract the tumor location and the tumor burden. Additionally, we can look at the features of the MRI images inside each of these two tumor parts.




jupyter 2.0-asi-deep_learning_brain_tumor_segmentation Last Checkpoint: Yesterday at 11:54 AM (autosaved)

Alexander S. Lundervold, 07.09.22

## Introduction

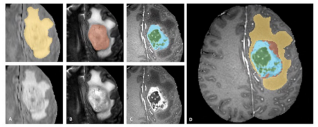
> This notebook is based on the `brats_segmentation_3d.ipynb` notebook from https://github.com/Project-MONAI/tutorials. You should consult the MONAI documentation for a more thorough introduction to the MONAI framework: https://monai.io.

This notebook will give a hands-on example of deep learning in practice. Along the way, we will see some of the deep learning concepts covered in the introduction to the workshop translated into code.

Our goal is to illustrate a possible approach to brain tumor segmentation, i.e., extracting meaningful tumor sub-regions. Having an accurate brain tumor segmentation system can lead to important imaging biomarkers such as f.ex. volumes and locations, and also provide regions of interest for extracting radiomics features. In another notebook, we take a practical look at radiomics.

## The data: The Brain Tumor Segmentation (BraTS) challenge

We will use the data from the BraTS 2016 and 2017 competitions, consisting of 750 multi-parametric MRI studies of patients with brain tumors. Each study consists of four MRI modalities, T1w, contrast-enhanced T1w, T2w, and FLAIR, and corresponding manual delineations of three tumor regions: necrotic tumor, active tumor, and edema.



### Sample images

It is a heterogenous data set. Here are some sample images:

fastMONAI

Overview
Vision core plot
Vision core
Vision data
Data augmentation
Custom loss functions
Vision metrics
Utils
Dataset information
External data
Tutorials
Single-label classification
Regression
Binary semantic segmentation
Multi-class semantic segmentation

# Overview

# fastMONAI

CI passing | Docs passing | PyPI version v0.1.3

A low-code Python-based open source deep learning library built on top of fastai, MONAI, and TorchIO.

fastMONAI simplifies the use of state-of-the-art deep learning techniques in 3D medical image analysis for solving classification, regression, and segmentation tasks. fastMONAI provides the users with functionalities to step through data loading, preprocessing, training, and result interpretations.

**Note:** This documentation is also available as interactive notebooks.

## Installing

On this page
Installing
From PyPI
From Github
Getting started
How to contribute
Citing fastMONAI
Report an issue

# Plan

What *is* deep learning?

How do you *do* deep learning?

Recent developments in generative AI

# Generative AI

Curated by @aaronsiim

Text-to-Image (T2I)

DALL·E 2 · Stable Diffusion · craiyon · lexica · Imagen · WOMBO · NightCafe · GauGAN2 · DeepAI · Jasper · artbreeder · Wonder · pixray-text2image · neural love · Omneky · alpaca · mage.space · KREA · Nyx gallery · ROSEBUD.AI · PhotoRoom

Text-to-Video (T2V)

runway · Fliki · synthesia · Meta AI · Google AI · Phenaki · CONTENDA

Text-to-Audio (T2A)

Play.ht · MURF.AI · RESAMBLE.AI · WELLSAID

Text-to-Text (T2T)

Simplified · Jasper · frase · grammarly · copy.ai · MarketMuse · AI21labs · InferKit · GooseAI · Writesonic · Ideas AI · copysmith · Flowrite · ideasby · text.cortex · OpenAI GPT-3 · Hypotenuse · Subtxt · WRITER · wordtune · LAIKA · Moonbeam · Bertha.AI · anyword

Text-to-Motion (T2M)

TREE.fm · MDM: Human Motion Diffusion Model

Text-to-Code (T2C)

replit Ghostwriter · GitHub Copilot · MUTABLE · Amazon CodeWhisperer

Text-to-NFT (T2N)

LensAI

Text-to-3D (T2D)

DreamFusion · CLIP-Mesh · GET3D

Audio-to-Text (A2T)

descript · AssemblyAI · Whisper

Audio-to-Audio (A2A)

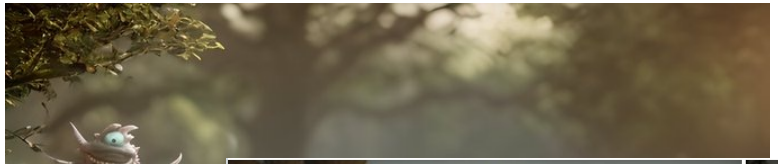
AudioLM · VOICEMOD

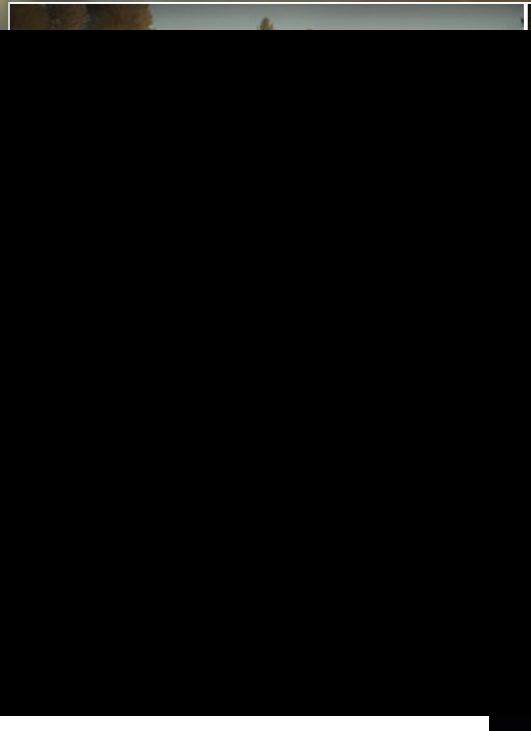
Brain-to-Text (B2T)

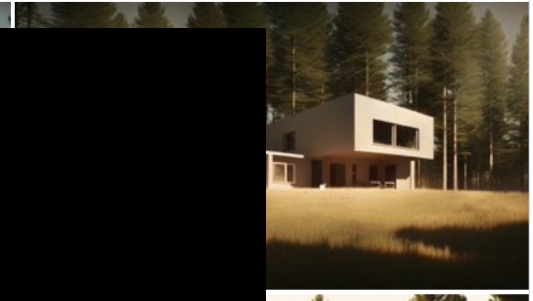
speech from brain · non-invasive brain recordings

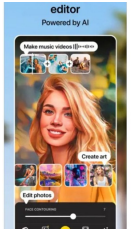
Image-to-Text (A2T)

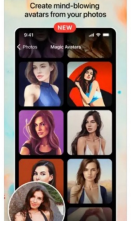
neural love · GPT-3 x Image Captions

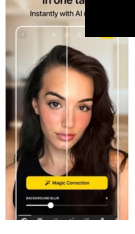


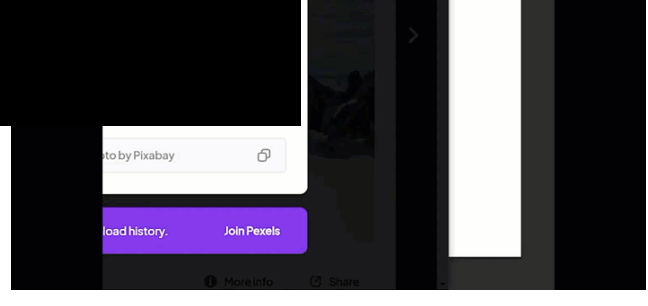






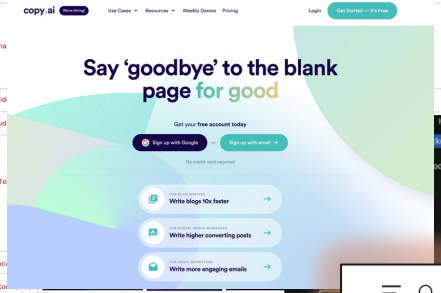



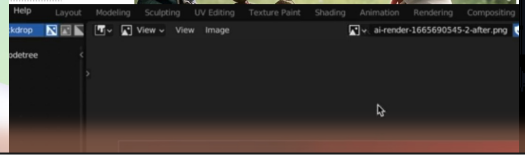



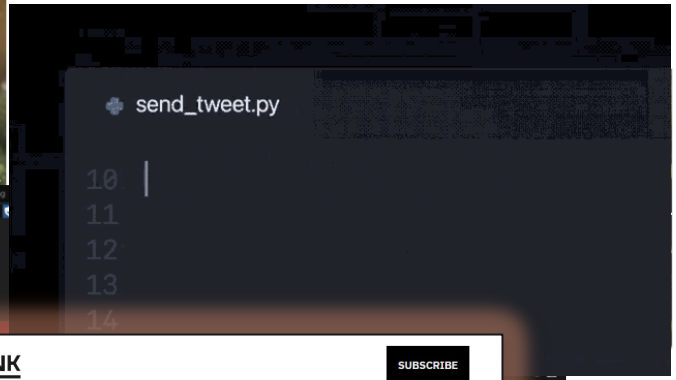



# Generative AI

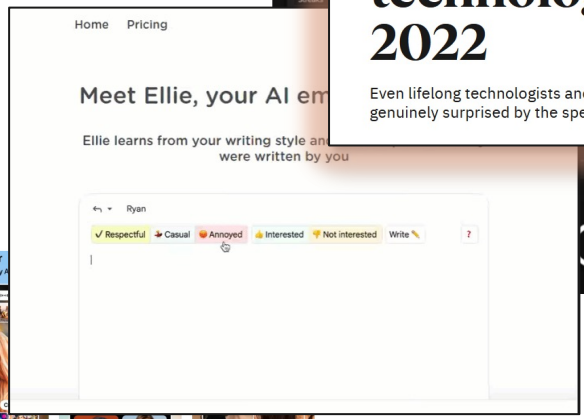
Curated by @aaronsiim











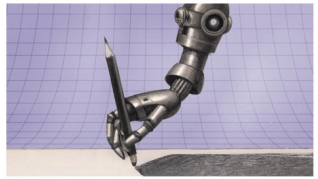
BIG THINK

SUBSCRIBE

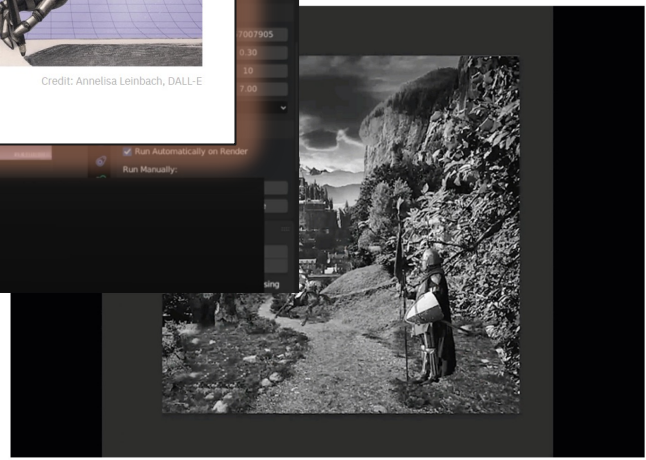
*THE PRESENT — DECEMBER 20, 2022*

# Generative AI: The technology of the year for 2022

Even lifelong technologists and AI researchers like myself were genuinely surprised by the speed and impact of generative AI.



Credit: Annelisa Leinbach, DALL-E

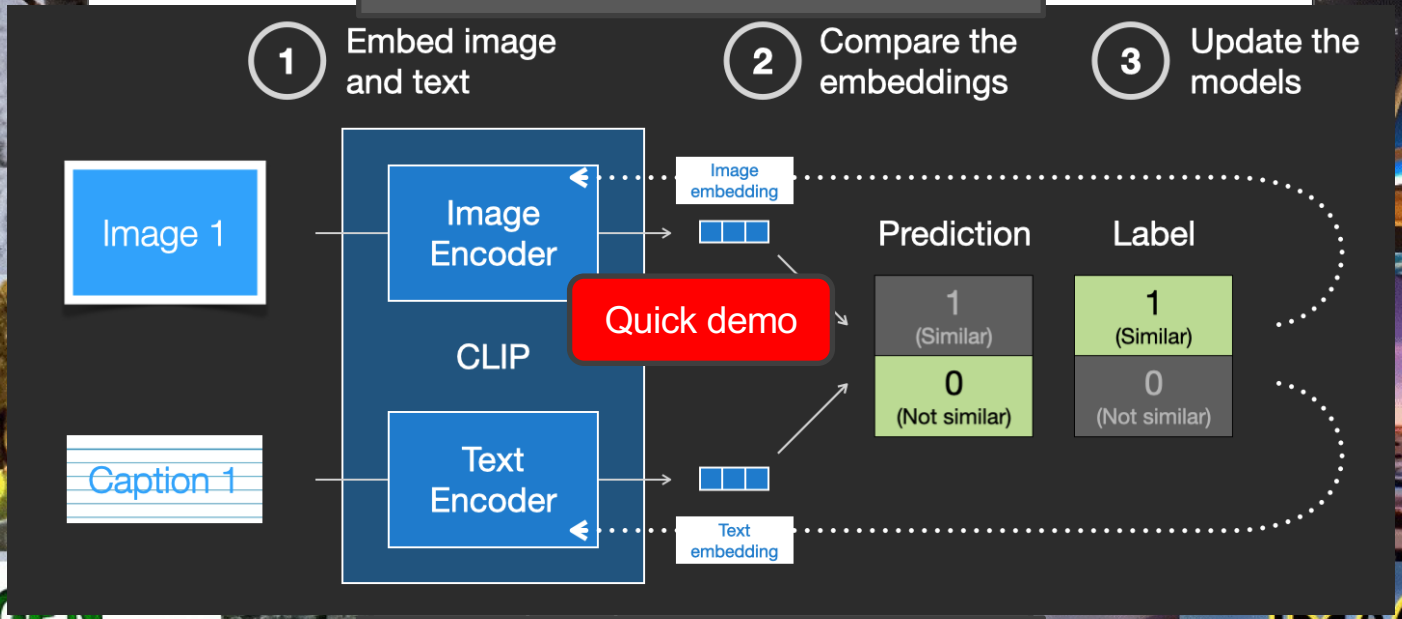



MAKE-A-VIDEO: TEXT-TO-VIDEO GENERATION WIT

Uriel ... Yin +

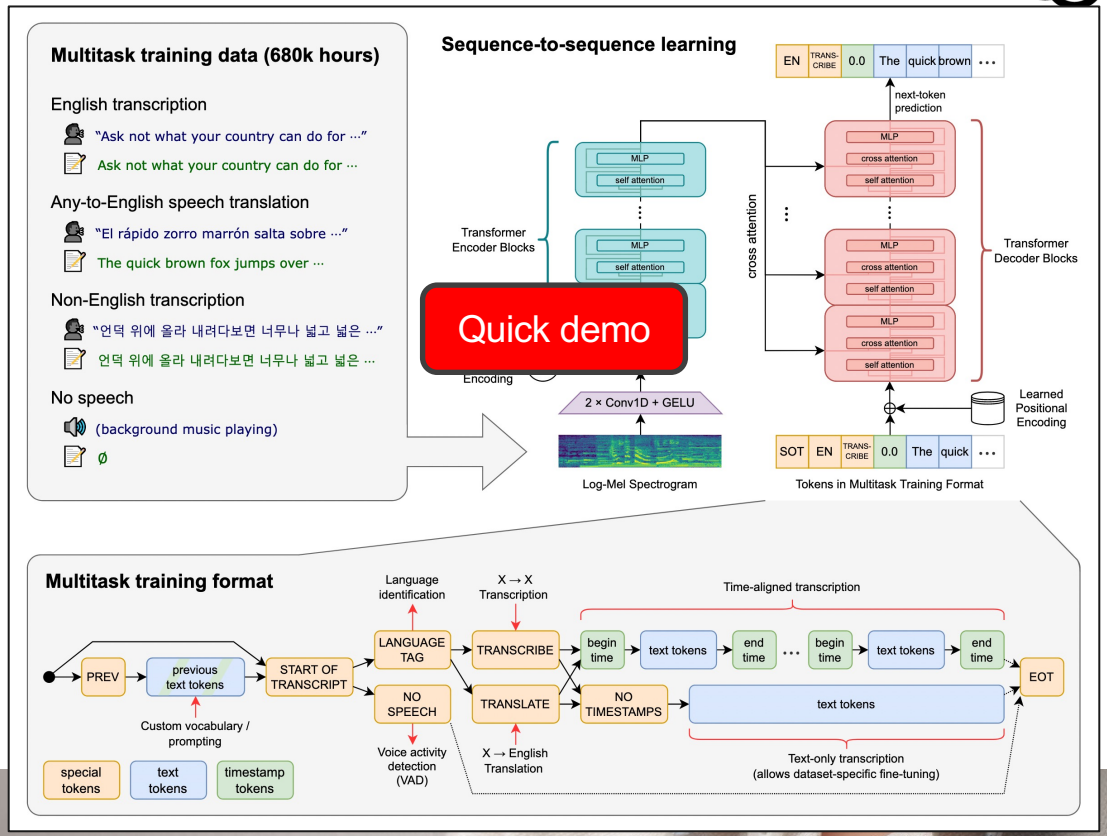
***Embeddings*** are core to these approaches.

Example from Stable Diffusion:

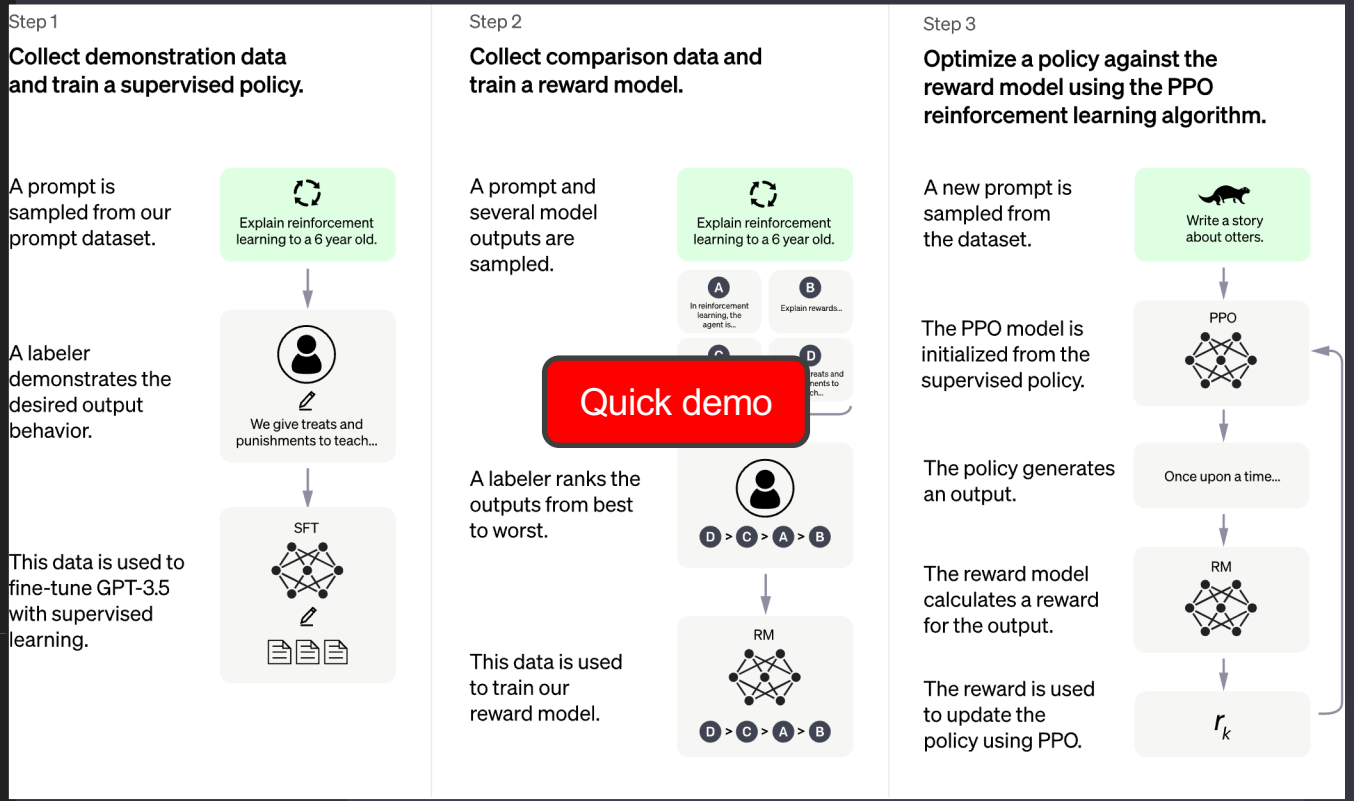



These text encodings are then used in the so-called ***diffusion process*** generating images from noise

# Introducing Whisper



ChatGPT



ChatGPT

en oppskrift

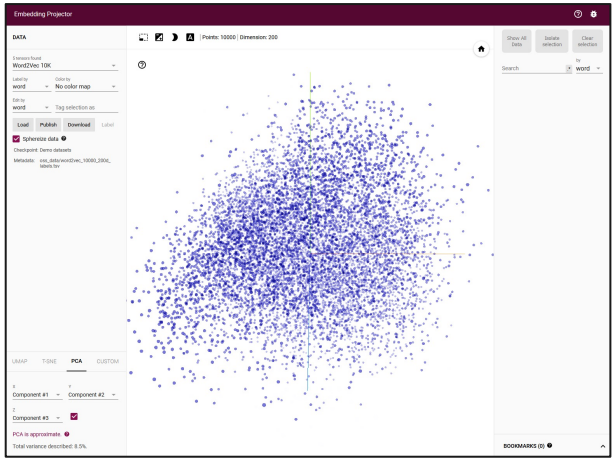
**korrekt**

**Mål**: Lage en chatbot som er **hjelpsom**, ~~**ærlig**~~, og **harmløs**

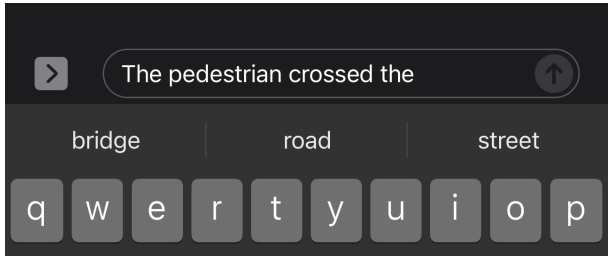
# ChatGPT en oppskrift

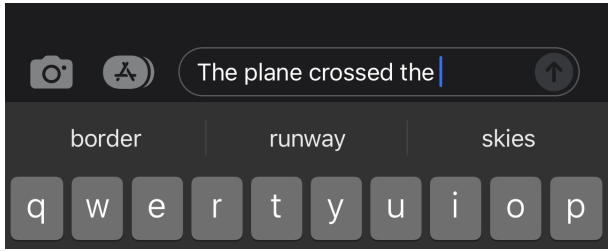
**Mål**: Lage en chatbot som er **hjelpsom**, **ærlig**, og **harmløs**

1 Start med GPT-3, en **large language model** (LLM)




http://projector.tensorflow.org/

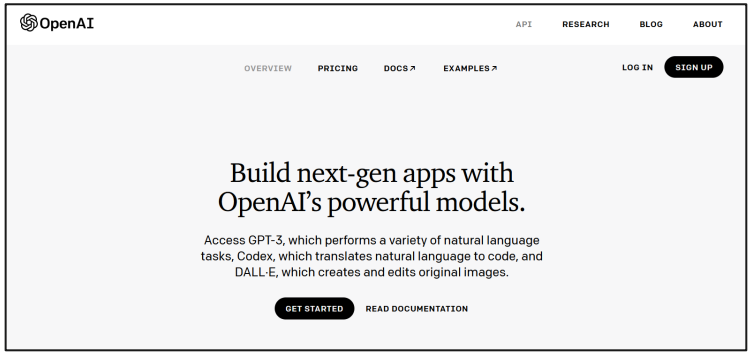


# ChatGPT en oppskrift

**Mål**: Lage en chatbot som er **hjelpsom**, **ærlig**, og **harmløs**

1. Start med GPT-3, en **large language model** (LLM)
2. Samle inn noen eksempler på spørsmål & svar. Start en treningsprosess (*fin-tuning* av GPT-3)
3. Generer flere svar på noen tusen spørsmål, be noen rangere svarene fra best til verst

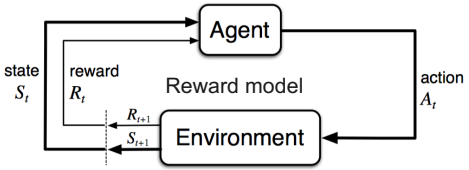



Stable diffusion v1.5. *"A group of people with computers in a foreign land, cartoon, matte painting,*

# ChatGPT en oppskrift

**Mål**: Lage en chatbot som er **hjelpsom**, **ærlig**, og **harmløs**

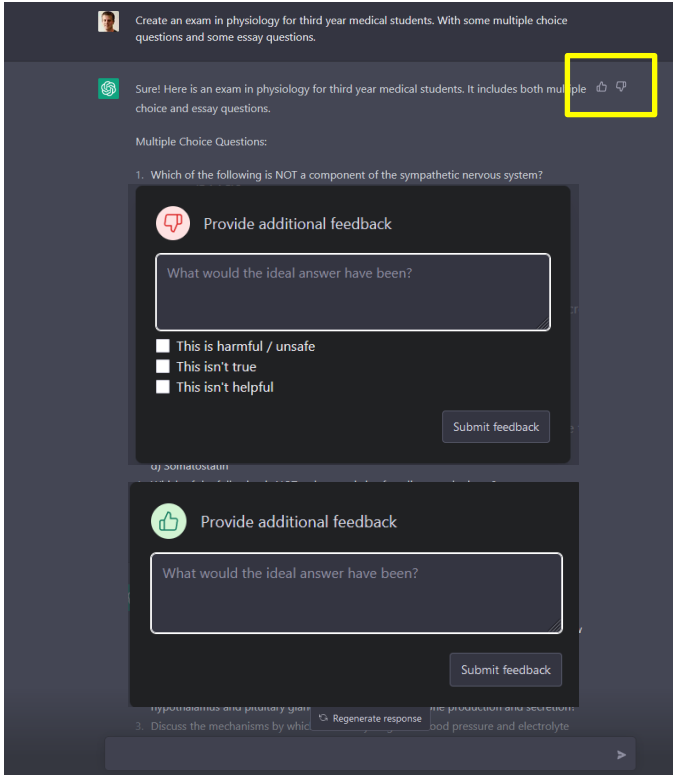
1. Start med GPT-3, en **large language model** (LLM)
2. Samle inn noen eksempler på spørsmål & svar. Start en treningsprosess (fin-tuning av GPT-3)
3. Generer flere svar på noen tusen spørsmål, be noen rangere svarene fra best til verst
4. Tren en annen modell til å **predikere rangeringen** menneskene ville gitt. Bruke denne til å trene GPT-modellen på noen millioner eksempler
5. Underveis, be personer gi rangering på eksempler der en estimerer at GPT-modellen kan lære mest mulig.




# ChatGPT en oppskrift

**Mål**: Lage en chatbot som er **hjelpsom**, **ærlig**, og **harmløs**

1. Start med GPT-3, en **large language model** (LLM)
2. Samle inn noen eksempler på spørsmål & svar. Start en treningsprosess (fin-tuning av GPT-3)
3. Generer flere svar på noen tusen spørsmål, be noen rangere svarene fra best til verst
4. Tren en annen modell til å **predikere rangeringen** menneskene ville gitt. Bruke denne til å trene GPT-modellen på noen millioner eksempler
5. Underveis, be personer gi rangering på eksempler der en estimerer at GPT-modellen kan lære mest mulig.
6. Gjør systemet offentlig tilgjengelig (“ChatGPT”) og bruk feedback fra brukere til å trene videre



# ChatGPT en oppskrift

**Mål**: Lage en chatbot som er **hjelpsom**, **ærlig**, og **harmløs**

1. Start med GPT-3, en **large language model** (LLM)
2. Samle inn noen eksempler på spørsmål & svar. Start en treningsprosess (fin-tuning av GPT-3)
3. Generer flere svar på noen tusen spørsmål, be noen rangere svarene fra best til verst
4. Tren en annen modell til å **predikere rangeringen** menneskene ville gitt. Bruke denne til å trene GPT-modellen på noen millioner eksempler
5. Underveis, be personer gi rangering på eksempler der en estimerer at GPT-modellen kan lære mest mulig.
6. Gjør systemet offentlig tilgjengelig (“ChatGPT”) og bruk feedback fra brukere til å trene videre